AlliedWare™ OS

How To | Configure Hardware Filters on AT-9900, x900-48, and x900-24 Series Switches

Introduction

The AT-9900, x900-48, and x900-24 series switches support a powerful hardware based packet-filtering facility.

These switches can filter on a range of Layer 2, Layer 3, and Layer 4 packet attributes, and perform a variety of different actions on the packets that match the filters.

Because the filters are hardware-based, they put no load on the CPU of the switch, and have no affect on the throughput of the switch. It is possible to configure over 1000 different filters, and still have complete wire speed throughput on the switch.

The following configuration methods are available:

1. To filter traffic across all ports on the switch, create dedicated hardware filters.
2. To filter traffic on a per-port basis, apply filtering actions to QoS flow groups or traffic classes.

This Note only describes method 1. Method 2 is described in *How To Configure Filtering Actions on QoS Flow Groups and Traffic Classes*, available from www.alliedtelesis.com/resources/literature/howto.aspx.

What information will you find in this document?

This document contains the following:

Introduction 1
Which products and software versions does this information apply to? 2
Creating dedicated hardware filters 3
Configuring packet classification 3
Configuring Layer 4 source and destination port number masks 4
Configuring "inner" parameters for nested VLANs 4
Creating hardware filters 5
The logic of the operation of the hardware filters 6
The effects of the action parameters 6
Combining hardware filters and QoS 7
How many filters can you create? 7
1. The filter rules table 7
Extra rules used when combining QoS and hardware filters 8
2. The profile (mask) 9
Are there enough bytes for your set of filters? 10
Some protocols also use filters, so use some of the length 11
How to see the current filter resource usage on the switch 12
Appendix A: How to use the layer 4 mask in classifiers 13
Example 1: ports 2000-2003 14
Points to remember 14
Example 2: ports 5004-5008 15
Example 3: ports 333-777 15

Which products and software versions does this information apply to?

- Products: AT-8948, AT-9900, x900-48, and x900-24 series
- Software versions: 2.7.3 and above

Hardware filters are also available on Layer 3 switches running the AlliedWare Plus OS. See the following How To Note:

- *How To Configure Hardware Filters on SwitchBlade x908, x900-12XT/S, and x900-24 Series Switches*

This Note is available from www.alliedtelesis.com/resources/literature/howto_plus.aspx.

Creating dedicated hardware filters

Before we get into the details of the filter creation, we need to look at the underlying packet classification process.

Configuring packet classification

Dedicated hardware filters and QoS use the same packet classification process.

The basic construct in the classification process is a classifier. The syntax for creating a classifier on the switch is:

```
CREate CLASSifier=rule-id
    [MACSaddr={macadd|ANY|DHCPSnooping}]
    [MACDaddr={macadd|ANY}][MACSMask=macadd][MACDMask=macadd]
    [MACType={L2Ucast|L2Mcast|L2Bcast|ANY}] [TPID={tpid|ANY}]
    [VLANPriority={0..7|ANY}] [VLAN={vlanname|1..<VIDMaxUser>|ANY}]
    [INNERTpid={tpid|ANY}] [INNERVLANPriority={0..7|ANY}]
    [INNERVLANId={vlanname|1..4094|ANY}]
    [ETHFormat={802.2-Tagged|802.2-Untagged|ETHII-Tagged|
    ETHII-Untagged|NETWARERAW-Tagged|Netwareraw-untagged|
    SNAP-Tagged|SNAP-Untagged|ANY}] [PROTocol={protocoltype|IP|IPV6|ANY}]
    [IPDScp={dscplist|ANY}] [IPTOs={0..7|ANY}]
    [IPSAddr={ipaddmask|ANY|DHCPSnooping}] [IPDAddr={ipaddmask|ANY}]
    [IPPRotocol={TCP|UDP|ICMp|IGMp|OSPf|ipprotocolnum|ANY}]
    [IPXDAddr={ipxadd|ANY}]
    [IPXDSocket={NCP|SAP|RIP|NNB|DIAg|NLSp|IPXwan|ipxsocketnum|ANY}]
    [IPXSSocket={NCP|SAP|RIP|NNB|DIAg|NLSp|IPXwan|ipxsocketnum|ANY}]
    [TCPSport={portid|port-range|ANY}] [TCPDport={portid|port-range|ANY}]
    [UDPSport={portid|port-range|ANY}] [UDPDport={portid|port-range|ANY}]
    [L4SMask=mask] [L4DMask=mask] [L5BYTE01=byteoffset,bytevalue[,bytemask]]
    [L5BYTE02=byteoffset,bytevalue[,bytemask]]
    ...
    [L5BYTE16=byteoffset,bytevalue[,bytemask]]
    [TCPFlags={{Urg|Ack|Rst|Syn|Fin}[,...]|ANY}]
    [ICmptype={Any|ECHORply|Unreachable|Quench|Redirect|ECHO|ADvertisement|
    Solicitation|TImeexceed|Parameter|TSTAMP|TSTAMPRply|INFOREQ|INFOREP|
    ADDRREQ|ADDRREP|NAMEREq|NAMERPly|icmp-type}]
    [ICMPCode={Any|FIlter|FRAGMent|FRAGReassm|HOSTComm|HOSTIsolated|HOSTPrec|
    HOSTREdirect|HOSTRTos|HOSTTos|HOSTUNKnown|HOSTUNReach|NETComm|
    NETREdirect|NETRTos|NETTos|NETUNKnown|NETUNReach|NOptr|POrtunreach|
    PREcedent|PROtunreach|PTrproblem|Sourceroute|Ttl|
    icmp-code}]
    [IGmptype={ANY|QUery|V1Report|DVmrp|PIMv1|CTRace|V2Report|V2Leave|
    MCTRACEResponse|MCTRACE|V3Report|MRAdvert|MRSolicit|MRTermination|igmp-
    type}]
    [EIPBYTE01=byteoffset,bytevalue[,bytemask]]
    [EIPBYTE02=byteoffset,bytevalue[,bytemask]]
    ...
    [EIPBYTE16=byteoffset,bytevalue[,bytemask]]
```

From this, it can be seen that there are a large number of different attributes upon which packets can be classified.

Most of these options are self-evident, but the following sections give more information about the L4 mask and the "inner" options. For information about the other options, see the *Generic Classifier* chapter of the Software Reference.

Configuring Layer 4 source and destination port number masks

A common filtering requirement is the ability to filter on a range of TCP or UDP port numbers. For example, we often want to be able to allow through all packets with a TCP destination port greater than 1024, as such packets are deemed to be replies coming back to sessions initiated from the other side of the switch.The **l4smask** and **l4dmask** parameters make it possible for a single classifier to match a whole range of port numbers.

These parameters take on HEX values, and are used in conjunction with the parameters **tcpsport**, **tcpdport**, **udpsport**, and **udpdport**. A range of port numbers matches the classifier if performing a logical AND with the mask would give the same result as performing a logical AND with the value specified in the corresponding **sport** or **dport** parameter.

Of course, this is not quite so convenient as being able to simply specify a range of decimal numbers. Often it can require multiple port/mask combinations to cover a particular range of numbers.

This maths of all this is described in detail in Appendix A of this How To Note—see page 13.

Note: The default value of each mask is FFFF. This means that if you specify a port number without specifying a mask, then the classifier matches only that one value of the port number. This is the same as specifying a port number and a mask of FFFF.

Configuring “inner” parameters for nested VLANs

The **tpid**, **innertpid**, **innervlanid**, and **innervlanpriority** parameters all apply to nested VLAN configuration. In this situation, the packets arriving at the core-facing port can have two VLAN tags configured on them.

- The **tpid** parameter matches on the first Tag Protocol Identifier field in the packet.
- The **innertpid** parameter matches on the TPID in the second 802.1Q tag in the packet.
- The **innervlanid** parameter matches on the tunnelled VLAN ID in the second 802.1Q tag in the packet.
- The **innervlanpriority** parameter matches on the 802.1P field in the second tag in the packet.

The following table shows where in the packet the inner and outer tags will be matched.

	Outer VLAN parameters (normal)	**Inner VLAN parameters**
Customer port	VLAN	1st tag
Core port	1st tag	2nd tag
Nested VLANs disabled	1st tag	2nd tag

Some important points to keep in mind while configuring the “inner” parameters are:

- When packets arrive at a customer port of a nested VLAN, the parameter **vlan** will match the VID of the nested VLAN that the port is a member of, which is just how this parameter normally operates.

- When packets arrive at a customer port of a nested VLAN, the "inner" parameters will match the attributes of the first tag in the packets. This is because when the packet is forwarded from the core port, that first tag will have become the inner tag. So, from the point of view of the nested VLAN, the tag that is on the packet when it arrives into the customer port is the inner tag.
- When nested VLANs are disabled, and "inner" parameters have been configured, these parameters will be applied as though all packets arriving at the switch were double tagged. In other words, there will be no attempt to make a distinction between "customer" and "core" ports. So, if the packets arriving at the switch are not double tagged, then the "inner" parameters will just match on whatever data happens to be in the packets at the position where an inner tag would have been.

  Therefore, when you disable nested VLANs, you should also remove the classifiers.
- When nested VLANs are being used, the parameters **tpid** and **vlanpriority** cannot be used in classifiers on filters applied to customer ports.
- If you attach the classifier to a number of ports, they will all be treated like core ports if at least one of the ports is a core port.

Creating hardware filters

Once you have created a classifier, create a filter. The filter uses the classifier, and specifies an action.

```
add switch hwfilter[=<filter-id>] classifier=<rule-id>
    action={copy|discard|forward|copy,discard|setl2qos}
```

Note that it is possible, but not required, to specify a ID number for the filter. If you do not specify an ID, then the filter is simply added to the end of the existing list of filters. However, if you want to actually insert a filter into a specific position in the list, then you can specify a filter ID. That way, the filter will be inserted at the position indicated by the filter-id value, and all the existing filters from that position and above will all move up one position.

For example, imagine you have the following set of filters:

```
add swi hwfilt class=1 action=xxx
add swi hwfilt class=4 action=xxx
add swi hwfilt class=3 action=xxx
add swi hwfilt class=6 action=xxx
```

Then, enter the following command:

```
add swi hwfilt=2 class=8
```

The new filter will be inserted at position 2 in the list. The previous filter #2 will become filter #3, the previous filter #3 will become filter #4, and the previous filter #4 will become filter #5:

```
add swi hwfilt class=1 action=xxx
add swi hwfilt class=8 action=xxx
add swi hwfilt class=4 action=xxx
add swi hwfilt class=3 action=xxx
add swi hwfilt class=6 action=xxx
```

The logic of the operation of the hardware filters

The operation of the filters follows the standard ACL logic: if a packet matches an filter, the comparison process stops and the action attached to the filter is performed. If a packet fails to match any of the filters, then the default action (forward) is taken.

Note: Hardware filters will act on packets that are destined for the switch itself (packets that would be passed up to the switch's own CPU) in exactly the same way as they act on packets that were destined to be forwarded directly by the switching chip.

The effects of the action parameters

Let us consider the effect of each the possible action keywords.

Action	What it does	When do you need this action?
discard	Drops the traffic.	Use this when the filtering policy is to disallow certain traffic flows.
forward	Forwards the traffic normally.	Use this when you want to discard a wide range of traffic, but still forward some small subset of traffic within that range.
copy	Forwards the traffic normally, and also sends a copy of each packet to the CPU.	Use this when you want software monitoring of a certain packet flow. If you want to log, or count, or output debug pertaining to a certain stream, then create a filter that matches the packets in the stream, and specify copy for the action.
copy,discard	Drops the traffic, but also sends a copy of each packet to the CPU.	Use this when you want software monitoring of a certain packet flow that is being dropped. If you want to log, count, or output debug pertaining to a certain disallowed stream, then create a filter that matches the packets in the stream, and specify copy,discard for the action.

setl2qos

Note that this action has the other parameters associated with it, as the following syntax shows:

```
add switch hwfilter[=<filter-id>] classifier=<rule-id> action=setl2qos
   [l2qosqueue=0..7] [priority=0..7] [bandwidthclass=1..3]
```

This action means you can use hardware filters to set the queue, 802.1p user priority or bandwidth class for packets.

There is an elaborate QoS mechanism available for allocating these values to packets, but this filter type provides a simple method if you do not require a full QoS configuration. The principle use for this filter action, though, is as a mechanism for elevating the probability of CPU reception for packets that you determine to be "important".

In heavily congested networks, data streams can sometimes use up all the available bandwidth of the CPU receive process. This increases the probability of losing infrequently-sent control or management packets, for example, routing protocol packets (BGP, OSPF, PIM, DVMRP) or STP packets. By creating an appropriate classifier and hardware filter, such packets can be given higher priority forwarding up to the CPU.

If you are using the filter to prioritise packets going up to the CPU, you only need to specify a value for the **l2qosqueue** parameter. The higher the value given to this parameter, the higher the priority the matching packets will be given in forwarding up to the CPU. It is possible to specify the **priority** and **bandwidthclass** parameters in this case, but they will have no effect, because the CPU ignores these parameters. The default value for the **l2qosqueue** parameter is 0.

The **priority** parameter specifies the 802.1p user priority with which to re-mark matching packets. The default is 0.

The **bandwidthclass** parameter specifies the bandwidth class (colour) to assign matching packets to. The default is 1 (green).

Combining hardware filters and QoS

The switch compares the packet with every hardware filter before it compares the packet with any QoS flow group. If the packet matches a hardware filter, the switch takes the action specified by that hardware filter and stops the comparison process. If a packet matches both a hardware filter and a QoS flow group, the packet only gets matched against the hardware filter. It bypasses the QoS process.

If the hardware filter actions include **discard**, then this is not a problem, because the packet was never going to get into the QoS system anyway (given that it was being discarded). But, if the hardware filter actions include **forward**, and the packet would also be matched by a QoS flow group, then this is a problem. The packet will not be matched by the QoS flow group, so the switch will not apply any intended QoS-based filtering, metering, queue redirection, etc to the packet. Instead the switch will forward the packet as if it belongs to the default traffic class for the port's QoS policy.

For this reason, we only recommend combining hardware filters and QoS if all your hardware filters result in traffic being dropped. For traffic that you want forwarded with QoS control, use QoS for both the filtering and the QoS functionality. Of course, you can also use QoS flow groups to drop traffic.

How many filters can you create?

The total number of filters that can be created is not an exact number, but depends on which fields the various filters are matching on. So, to understand how to work out whether the set of filters you are creating might run out of space, it is necessary to understand the way in which the filters operate in the switch hardware.

There are two items within the switch hardware which set limits on the number of filters that can be created: the filter rules table and the profile (mask).

Hardware filters and QoS share the same filter rules table and mask.

1. The filter rules table

One item that sets a limit on the number of filters is the table that contains the list of filter rules. This has a strict limit of 1024 entries. Entries get made when:

- You create a hardware filter.
- You use QoS to apply a classifier to a port.

Extra rules used when combining QoS and hardware filters

In fact, QoS can cause the limit on the number of hardware filters to be reduced rather more radically than might be initially evident. To see why this is, we have to understand a bit more about how the rule table is used. When a packet is to be compared against rules in the rule table, the comparison does not **have** to start at the top of the table—it can start at other points in the table. The decision as to the starting point for any particular packet is made on the basis of the packet's ingress port. When no QoS policies have been configured on the switch, and only hardware filters have been configured, it is convenient and simple to have the rule comparison process for **all** packets start at the top of the rule table and run to the last non-null entry in the table, regardless of the packet's ingress port. This is because hardware filters on the AT-9900 and x900 series switches are not ingress-port specific.

Therefore, when only hardware filters have been configured on the switch, **all** rule comparisons start at the first rule in the rule table, irrespective of the packet's ingress port.

Port	Start
1	1
2	1
3	1
4	1
5	1
...	...
...	...
52	1

Table that maps ingress port to the starting point of the rule comparison process

1	Rule 1
2	Rule 2
3	Rule 3
4	Rule 4
	Empty

Rule table

However, QoS policies are ingress-port specific. Different policies can be configured on different ports. So, the rules for allocating packet to flow groups can differ from port to port. Hence, QoS can result in the rule table containing different sets of rules for different ports.

This means that for the purposes of QoS, the decision that dictates the starting point of the rule comparison process, depending on ingress port, must result in different start values for different ingress ports. But, the problem is that the hardware filtering must use the exact same decision process. So we end up with a conflict of interests—the hardware filter process wants to run every packet through the same set of rules, but QoS wants to use different sets of rules for different packets, depending on the packet's ingress port. But it is not possible to make a single ingress-port-to-rule-table-starting-point decision process fulfil these two desires both at the same time.

The solution to this problem is as follows. As soon as a QoS policy is configured, which requires the creation of a set of rules specific just to the ports in that policy, then a full copy of the hardware filter rules is also added to this set of QoS rules. So, for packets entering the switch via one of the ports in the QoS policy, the hardware filter rule lookup process is actually carried out on this new copy of the hardware filter rules.

The following figure shows the copies of these rules.

Port	Start
1	1
2	1
3	1
4	9
5	9
6	1
...	...
...	...
52	1

Table that maps ingress port to the starting point of the rule comparison process

1	Rule 1
2	Rule 2
3	Rule 3
4	Rule 4
5	
6	
7	
8	
9	Copy of rule 1
10	Copy of rule 2
11	Copy of rule 3
12	Copy of rule 4
13	QoS rule #1
14	QoS rule #2

Rule table

When a QoS policy has been applied to ports 4 and 5, all the hardware filter rules have to be replicated further down in the rule table, and the QoS-specific rules added to the table below this copy of the hardware filter rules. For ports 4 and 5, the rule comparison process starts at entry 9 in the rule table, not at entry 1.

The entries 5-8 in the table have been left blank because separate sets of rules in the rule table must begin at an 8-entry boundary.

So, if there are several QoS policies configured on the switch, then there will be several copies of the hardware filter rules within the rule table. This, of course, can significantly reduce the maximum number of hardware filters that can be created.

Also, the protocols that use filters (see page 11) create at least one entry each.

2. The profile (mask)

The other item that affects the number of filters you can create is called the profile. Conceptually, this is a 16-byte mask that decides which set of bytes should be extracted from a packet as it enters the filtering process, to be compared against all the hardware filter and QoS classifiers. Hardware filters and QoS share a single mask.

In effect, the mask is the sum of all the individual bytes required for each individual classifier parameter. The number of bytes required by each classifier parameter depends on what fields it maps on. For example:

source MAC address—6 bytes

destination MAC address—6 bytes

Protocol type—2 bytes

Ethernet format—2 bytes

VLAN ID—2 bytes

IP protocol type (TCP, UDP, etc)—1 byte

source IP address—4 bytes

destination IP address—4 bytes

TCP port number—2 bytes

UDP port number—2 bytes

DSCP—1 byte

For example, if you make a hardware filter that matches on destination IP address and source TCP port, this adds 7 bytes to the mask:
1 byte for the IP protocol field (to indicate TCP)
4 bytes for the destination IP address
2 bytes for the source TCP port number.

If you next make a hardware filter that matches on source MAC address, this adds 6 more bytes to the mask.

If you next make a QoS flow group with a classifier that matches on destination IP address (4 bytes) and DSCP (1 byte), this adds 1 more byte to the mask, for the DSCP. It does not add 4 more bytes for the destination IP address because the switch already matches on that field.

If you next make a hardware filter that matches on source IP address and source TCP port, then that does not change the mask, because the switch already matches on those fields.

If you next make a hardware filter that matches on source UDP port, this also does not add any length to the mask, because it shares the same 2 bytes as the source TCP port. However, if you next make a hardware filter that matches on *destination* TCP or UDP port, that uses another 2 bytes.

Are there enough bytes for your set of filters?

Of course, the mask cannot increase without limit—it has a maximum size of 16 bytes.

When it reaches the 16-byte limit, no more classifiers can be used that would cause the mask to increase in size. The switch can still accept classifiers that use fields that have already been included in the mask.

There is no particular number of hardware filters or QoS flow groups that will cause the mask to reach its 16-byte limit—it could happen after a few filters, or you might be able to create hundreds of filters without the mask reaching its limit.

So to determine whether you will have enough filter length, look at the fields you want to filter, determine the number of bytes for each field, and sum up **the total number of bytes.** If that number is less than 16, there is enough filter length. Don’t forget to count TCP and UDP source port as a single field, and likewise to count TCP and UDP destination port as a single field.

Okay length For example, this set of filters would work:

source MAC address
source UDP port
destination IP address + destination TCP port

The total number of bytes for the switch to check in a packet would be:

source MAC address + IP protocol type + source TCP/UDP port +
destination IP address + destination TCP/UDP port =
6 + 1 + 2 + 4 + 2 = 15 bytes

Too long But this set of filters would not work:

source MAC address
destination MAC address
destination IP address + destination TCP port

The total number of bytes for the switch to check in a packet would be:

source MAC address + destination MAC address + IP protocol type +
destination IP address + destination TCP/UDP port =
6 + 6 + 1 + 4 + 2 = 19 bytes

Some protocols also use filters, so use some of the length

The following protocols use filters, and therefore use up some of the available profile length and filter entries:

EPSR EPSR matches on VLAN ID, which uses 2 bytes. EPSR is disabled by default.

IGMP snooping IGMP snooping matches on the IP protocol type field (to identify IGMP packets and send them to the CPU). This uses 1 byte. IGMP snooping is enabled by default.

DHCP snooping DHCP snooping matches on the IP protocol type field (1 byte) and the source and destination UDP ports (2 bytes each). Therefore, it uses 5 bytes in total. DHCP snooping is disabled by default.

MLD snooping MLD snooping matches on the IPv6 router alert option and its value (2 bytes). MLD snooping is enabled by default. If you are not using IPv6, you can turn off MLD snooping with the command **disable mldsnooping**.

How to see the current filter resource usage on the switch

The **show switch** command outputs a number of counters that display the current usage of filtering resources. A typical output from this command, and a discussion of each of the values it outputs, is shown below:

Command output	**Description**

`Traffic Control Unit,hardware`
`resource usage:`
`Total system rule space ... 2048`
	Total number of classifiers/filter rules available in the system. This is the sum of the rules available on the base system and the rules available on the IPv6 accelerator.

`Total number of rules used .... 8`	Number of classifiers currently being used
`Total rule space usage ........ 24`	Number of rules reserved, accounting for block size of 8. Even though there are only 8 rules in use, there have actually been 3 blocks of 8 rules allocated from the rule table, as the rules in the rule table must be allocated in blocks of 8. The blocks are: ● one block on the base system for packets arriving into the switch via port 1 (which has had a QoS policy applied to it) ● one block on the base system for packets arriving in via any other port ● one block allocated on the IPv6 accelerator.

`Number of rules per application:`
 	
`MLD Snooping ................ 4`
`Accel. Card(IPv6) ........... 1`
`Switch HwFilter ............. 2`
`QOS ......................... 1`
	Splitting the rule allocation out on a per-application basis:
	● 2 rules on port 1 for MLD, 2 for MLD on all the other ports
	● 1 default rule in the IPv6 card
	● 1 hardware filter rule for port 1, and one for all the other ports
	● 1 QoS rule for port 1

`Total number of actions ... 1024`	Total number of actions available for hardware filters or QoS
`Number of actions used ........ 10`	8 actions in use by the 8 rules, and 2 default actions (for packets that match no rules)

`Device Resource, device #0:`
`Number of rules used ........ 7`
`Rule space usage ............ 16`
	Resource being used by the first device, which is the base board
	● Same as the 8 above, except for the one that is on the IPv6 card
	● Two 8-rule blocks have been allocated on the base system

`Number of rules per application:`
`MLD Snooping .............. 4`
`Switch HwFilter ........... 2`
`QOS ....................... 1`
`Device rule space limit .. 1024`
	Splitting the rule allocation out on a per-application basis
	● 2 rules on port 1 for MLD and 2 for MLD on all the other ports
	● 1 hardware filter rule for port 1, and one for all the other ports
	● 1 QoS rule for port 1
	● Total number of rules in the rule table on the base system

Command output	Description

`Profile #1:`
 	
`IPv4 bytes used ......... 3 of 16`
`Other-Eth bytes used .... 5 of 16`
	Profile used to match on packets
	● Number of bytes being used in the profile for matching IPv4 packets
	● Number of bytes being used in the profile for matching non-IPv4 ethernet packets

`Device Resource, device #1:`
 	
`Number of rules used ........ 1`
`Rule space usage ............ 8`
	Resources used by device number 2 - accelerator card
	● 1 default rule in the IPv6 card
	● One 8-rule block has been allocated in the rule table on the accelerator card

`Number of rules per application:`
`Accel. Card(IPv6) ......... 1`
`Device rule space limit ... 1024`
	Splitting the rule allocation out on a per-application basis
	● 1 default IPv6 rule
	● Total number of rules in the rule table on the accelerator card

`Profile Usage:`
`Profile #1:`
 	
`IPv4 bytes used ......... 0 of 16`
`Other-Eth bytes used .... 6 of 16`
	Profile used to match on packets
	● Number of bytes being used in the profile for matching IPv4 packets
	● Number of bytes being used in the profile for matching non-IPv4 ethernet packets

Appendix A: How to use the layer 4 mask in classifiers

This section describes the use of L4 mask in classifiers and gives some examples on L4 masks.

The way that L4 masks work is similar to IP subnet masks. You need to be familiar with the binary system to set the right mask for your need.

The L4 mask is a 2-byte hexadecimal number, the base-16 number system, which consists of 16 unique symbols: the numbers 0 to 9 and the letters A to F.

For example, if we want to set our UDP destination port to 2000:

2000	=	11111010000 (in binary)
2000	=	07D0 (in hexadecimal)
The default mask	=	FF FF which is 11111111 11111111 (in binary)

Applying a L4 mask to an UDP/TCP port allows you to identify the constant and variable parts of the port number. The constant bits are represented by the 1s in the mask, and the variable bits are represented by the 0s. Performing a bitwise logical AND operation between the port number and the L4 mask results in the first port number of the range.

Note: The logical AND operation compares 2 bits and if they are both "1", then the result is "1", otherwise, the result is "0".

Let's look at some examples.

Example 1: ports 2000-2003

Let's say we want to have a UDP port range of 2000-2003, then the mask we need to have is:

```
2000      =     00010011 10001100
2001      =     00010011 10001101
2002      =     00010011 10001110
2003      =     00010011 10001111
```

The changed bits from 2000-2003 are bolded. We must now write a L4 mask which will meet these requirements. The easiest way to do is, we must set the changed bits (between 2000 and 2003) in the mask to 0. In our example, they are the last 2 bits. So our mask should be:

```
2000        =     00010011 10001100
2001        =     00010011 10001101
2002        =     00010011 10001110
2003        =     00010011 10001111
L4 Mask     =     11111111 11111100
```

We must convert the binary number of 11111111 11111100 to hex, which ends in **FF FC**.

The classifier for UDP destination ports between 2000-2003 should be:

```
Create class=1 udpdp=2000 l4dmask=FFFC
```

Points to remember

In our first example we choose a starting port number in which the last 2 bits were 0 and also choose the number of the ports as 4 (power of 2) to simplify the example.

Before going into the complex examples, there are some points to remember for the L4 mask calculation:

- if the beginning port is an odd number (last bit 1), to cover a range of ports, you will need an extra 1 classifier compared to the even-beginning ports.
- you can easily calculate the total number of ports in a mask by using the formula 2^x (where x is the number of the 0's at the end of the mask). For example, a mask of 1111111111111000 will cover a range of $2^3 = 8$ ports.
- Divide the total number of the ports you want to cover into a sum of powers of 2. For example, a range of 77 ports could be divided into:

  64 + 8 + 4 + 1 =77

This shows us that a group of 77 ports could be covered by a minimum of 4 classifiers.

Example 2: ports 5004-5008

In some more complex situations, we may need more than one classifier to cover all the range we want to. Let's take UDP destination ports between 5004-5008

```
5004     =     00010011 10001100
5005     =     00010011 10001101
5006     =     00010011 10001110
5007     =     00010011 10001111
5008     =     00010011 10010000
```

According to the bolded bits, we may think that the changed bits are the last 5 bits so the mask should be 11111111 11100000.

But remember that if we set the last 5 bits to 0, the mask will cover 2^5 = 32 ports. But we want to cover only 5 ports, so let's divide 5 into 4+1.

```
5004     =     00010011 10001100
5005     =     00010011 10001101
5006     =     00010011 10001110       4 ports
5007     =     00010011 10001111
                                       +
5008     =     00010011 10010000       1 port
```

Now it is really easy to write the classifiers!

```
create class=1 udpdp=5004 l4dmask=FFFC
create class=2 udpdp=5008
add swi hwfilt class=1,2 action=drop
```

Example 3: ports 333-777

A more complex situation, let's try to write the classifiers for UDP ports between 333-777. As we are trying to get rid of odd numbers in the beginning of our port range, we will prefer to write classifiers for single port number for 333.

- 777-334+1 = 444 total number of port (+1 for including 333)
- 256 + 128 + 32 + 16 + 8 + 4 = 444 (7 classifiers)
- 7 classifiers + 1 classifier (for 333) = 8 classifiers

Now the question is how to locate these blocks. The short-cut to do that is, first, find the position of the biggest block, which is 256 for our example. 256 should fit into one of the following blocks:

```
0       -   255
256     -   512
512     -   767
768     -   1024
...
65280   -   65536
```

So our biggest block fits into the range 512-767.

The next second biggest block is 128 in our example … it should fit into 384-511.

```
...
256     -  383
384     -  511
...
...
```

With these 2 blocks, we cover from 384-767. If we keep repeating the same procedure for the other blocks, we get the commands in the following table. In some of the cases, the blocks need to be divided into smaller blocks. In our example, instead of having a single block of 4, we used 2 x 2 blocks: one at the start (classifier 2) and one at the end (classifier 8).

Port range	Number of ports	Command
333	1	create class=1 udpdport=333
334-335	2	create class=2 udpdport=334 l4dmask=FFFE
336-351	16	create class=3 udpdport=336 l4dmask=FFF0
352-383	32	create class=4 udpdport=352 l4dmask=FFE0
384-511	128	create class=5 udpdport=384 l4dmask=FF80
512-767	256	create class=6 udpdport=512 l4dmask=FF00
768-775	8	create class=7 udpdport=768 l4dmask=FFF8
776-777	2	create class=8 udpdport=776 l4dmask=FFFE

L4 Mask	Number of ports
FFFF	1
FFFE	2
FFFC	4
FFF8	8
FFF0	16
FFEO	32
FFCO	64
FF80	128
FF00	256
FE00	512
FC00	1,024
F800	2,048
F000	4,096
E000	8,192
C000	16,384
8000	32,768
0000	65,536

The following table shows the port ranges for the largest blocks.

L4 mask:	**FC00**	**F800**	**F000**	**E000**	**C000**	**8000**	**0000**
number of ports:	**1024**	**2048**	**4096**	**8192**	**16384**	**32768**	**65536**
	0	0	0	0	0	0	0
	1024	2048	4096	8192	16384	32768	65536
	2048	4096	8192	16384	32768	65536	
	3072	6144	12288	24576	49152		
	4096	8192	16384	32768	65536		
	5120	10240	20480	40960			
	6144	12288	24576	49152			
	7168	14336	28672	57344			
	8192	16384	32768	65536			
	9216	18432	36864				
	10240	20480	40960				
	11264	22528	45056				
	12288	24576	49152				
	13312	26624	53248				
	14336	28672	57344				
	15360	30720	61440				
	16384	32768	65536				
	17408	34816					
	18432	36864					
	19456	38912					
	20480	40960					
	21504	43008					
	22528	45056					
	23552	47104					
	24576	49152					
	25600	51200					
	26624	53248					
	27648	55296					
	28672	57344					
	29696	59392					
	30720	61440					
	31744	63488					
	32768	65536					
	33792						
	34816						
	35840						
	36864						
	37888						
	38912						
	39936						
	40960						
	41984						
	43008						
	44032						
	45056						
	46080						
	47104						
	48128						

L4 mask:	FC00	F800	F000	E000	C000	8000	0000
number of ports:	1024	2048	4096	8192	16384	32768	65536
	49152						
	50176						
	51200						
	52224						
	53248						
	54272						
	55296						
	56320						
	57344						
	58368						
	59392						
	60416						
	61440						
	62464						
	63488						
	64512						
	65536						

C613-16058-00 REV C